CSE 160 Winter 2020: Midterm Exam

(closed book, closed notes, no calculators)

Instructions: This exam is closed book, closed notes. You have 50 minutes to complete it. It contains 10 questions and 9 pages (including this one), totaling 100 points. Before you start, please check your copy to make sure it is complete. Turn in all 9 pages of the exam, together, when you are finished. When time has been called you must put down your pencil and stop writing. Points will be deducted from your score if you are writing after time has been called. You should only use parts of Python that have been covered in the class so far.

Good Luck!

Total: 100 points. Time: 50 minutes.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Points Possible</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>14</td>
</tr>
<tr>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>10</td>
<td>24</td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
</tr>
</tbody>
</table>
1) [6 pts] For each of the if statements below, write the output when \( x = 10, \ x = 30, \) and \( x = 200 \) in the table below. If there is no output then write “NO OUTPUT”.

a)  
if \( x > 20 \):
   if \( x < 50 \):
      print("line 1")
   else:
      if \( x < 500 \):
         print("line 2")
      print("line 3")

b)  
if \( x > 100 \):
    print("line 4")
elif \( x <= 40 \):
    print("line 5")
if \( x > 10 \):
    print("line 6")

<table>
<thead>
<tr>
<th></th>
<th>( x = 10 )</th>
<th>( x = 30 )</th>
<th>( x = 200 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code a)</td>
<td>NO OUTPUT</td>
<td>line 1 line 3</td>
<td>line 2 line 3</td>
</tr>
<tr>
<td>Code b)</td>
<td>line 5</td>
<td>line 5 line 6</td>
<td>line 4 line 6</td>
</tr>
</tbody>
</table>

2) [4 pts] Write the output of the code below in the box here:

```python
sum = 0
for x in range(5, 2, -1):
    for y in range(x):
        sum = sum + y
print("sum:", sum)
```

**MY ANSWER:**

```
sum: 19
```
3) [4 pts] What output is produced after running the following piece of code?

```python
A = [5, 7]
B = list(A)
C = A
D = A.append(8)
B.insert(1, 9)
C[1] = 4

print(A)
print(B)
print(C)
print(D)
```

**MY ANSWER:**

```
[5, 4, 8]
[5, 9, 7]
[5, 4, 8]
None
```

4) [8 pts] Suppose we have the following list:

```python
happy = [2, [1, [4, 3, 0], 5, 2], [8], 6, [9, 7]]
```

Write the result of the following expressions. If an error is thrown, briefly describe the error.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) happy[1]</td>
<td>[1, [4, 3, 0], 5, 2]</td>
</tr>
<tr>
<td>b) happy[2]</td>
<td>[8]</td>
</tr>
<tr>
<td>c) [9] in happy</td>
<td>False</td>
</tr>
<tr>
<td>d) 5 in happy[1]</td>
<td>True</td>
</tr>
<tr>
<td>e) happy[1][1][0]</td>
<td>4</td>
</tr>
<tr>
<td>f) happy[2][0]</td>
<td>8</td>
</tr>
<tr>
<td>g) happy[2:4]</td>
<td>[[8], 6]</td>
</tr>
<tr>
<td>h) len(happy)</td>
<td>5</td>
</tr>
</tbody>
</table>
5) [12 pts] For each of the following statements, show what is printed. If nothing is printed then write “NO OUTPUT”.

```python
x = 5.0
a = 2

def foo(a, d):
    j = a
    for i in range(a):
        print("hello")
        j = j - 1
    print("world", j)
    print("in foo", dog(j + 1, d))

def dog(x, y):
    print("in dog", y)
    y = x + y
    return y

def cat(a):
    a = a * a
    print("in cat", a)
    return a / 2

a) print(dog(a, x))

    in dog 5.0
    7.0

b) print(cat(x))

    in cat 25.0
    12.5

c) print(foo(a, x))

    hello
    hello
    world 0
    in dog 5.0
    in foo 6.0
    None
```
6) [8 pts] Given the following variables that have been defined. What is the result and type of the following expressions (if it is an Error indicate that, and leave result and type blank):

\[
\begin{align*}
a &= 2 \\
b &= 100 \\
c &= 3.0 \\
snail &= \text{"True"} \\
rain &= a < b
\end{align*}
\]

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
<th>Type</th>
<th>Error? (yes/no)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \frac{b}{a} )</td>
<td>50.0</td>
<td>float</td>
<td>no</td>
</tr>
<tr>
<td>( c \leq a )</td>
<td>False</td>
<td>boolean</td>
<td>no</td>
</tr>
<tr>
<td>( snail[a] )</td>
<td>\text{&quot;u&quot;}</td>
<td>string</td>
<td>no</td>
</tr>
<tr>
<td>( \text{rain and } b &gt; c )</td>
<td>True</td>
<td>boolean</td>
<td>no</td>
</tr>
</tbody>
</table>

7) [8 pts] What is the output of the following code? If the code has an error write “Error”.

\[
\begin{align*}
\text{big} &= \{0, 1, 2, 3, 4\} \\
\text{small} &= \{4, 5\}
\end{align*}
\]

a) print(big & small)

\{4\}

b) print(small | big)

\{0, 1, 2, 3, 4, 5\}

c) element = big.add(4)
   print(element)
   print(big)

   None
   \{0, 1, 2, 3, 4\}

d) print((big - small).remove(5)))

   KeyError: 5
8) [14 pts] a) **Draw** the entire environment, including all active environment frames and all user-defined values, **at the moment that the MINUS OPERATION IS performed**. Feel free to draw out the entire environment, but be sure to CLEARLY indicate what will exist at the moment the MINUS operation is performed (e.g. cross out frames that no longer exist).

b) When finished executing, **what is printed out by this code?**

```
MY ANSWER: 16
```

c) **How many different stack frames** (environment frames) are active **when the minus operation is performed**? (Hint: The global frame counts as one frame.)

```
MY ANSWER: 2
```

---

```python
x = 10
y = 300

def mouse(x):
    temp = bear(x) + 2
    return ape(x, temp) - temp

def ape(y, x):
    return y + bear(x)

def bear(x):
    return x + 3

print(mouse(bear(x)))
```

**MY ANSWER:**

[Diagram of the execution environment with stack frames and active frames at the moment of MINUS operation.]
9) [12 pts] A matrix is a list of lists, similar to our `pixel_grids` from HW3, except a matrix should be a square (have the same number of rows as columns). For example,

\[
A = \begin{bmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12 \\
13 & 14 & 15 & 16
\end{bmatrix}
\]

`trace(matrix)` should return the sum of all the entries along the diagonal, from the upper left corner down to the lower right corner. For matrix `A` above, the trace is defined as:

\[
trace(A) = 1 + 6 + 11 + 16 = 34
\]

If `matrix` is not a square (the number of rows is not the same as the number of columns), `trace` should return `None`. You may assume each row in the matrix has the same number of entries. You may assume that `matrix` contains at least one row and one column.

```python
def trace(matrix):
    # Write your code here
    if (len(matrix) == len(matrix[0])):
        diag_sum = 0
        for row in range(len(matrix)):
            diag_sum += matrix[row][row]
        return diag_sum
    else:
        return None  # Not required
```
10) [24 points total]

a) [8 pts] Write a function adjusted_pixel(pixel, val) where pixel is an integer representing a single pixel value and val is an integer representing the amount that this pixel should be adjusted. adjusted_pixel returns pixel + val. In addition:

- If pixel + val is less than 0 then it just returns 0.
- If pixel + val is greater than 255, then it just returns 255.

You may assume pixel and val are both integers and only their sum needs to be checked to see if it is within the valid range for pixels: (as in HW3, the valid range is: 0 to 255).

The call: adjusted_pixel(200, 5) would return: 205
The call: adjusted_pixel(250, 60) would return: 255
The call: adjusted_pixel(100, -6) would return: 94
The call: adjusted_pixel(10, -60) would return: 0
The call: adjusted_pixel(300, -100) would return: 200

```python
def adjusted_pixel(pixel, val):
    # Write your code here

    new_pixel = pixel + val
    if new_pixel > 255:
        new_pixel = 255
    elif new_pixel < 0:
        new_pixel = 0
    return new_pixel
```
10) b) [16 pts] Write a function `adjusted_grid(pixel_grid, amount)` that takes a pixel_grid as described in HW 3 as an argument and returns a new pixel_grid, with all of its pixels adjusted by amount. As in HW3, the valid range for pixels is: 0 to 255. So any pixel + amount that goes over 255 will be capped at 255 and any pixel + amount that goes below 0 will be set to 0. For example:

`adjusted_grid([[1, 2], [3, 4], [7, 2]], 3)` returns:

```
[[4, 5], [6, 7], [10, 5]]
```

`adjusted_grid([[8, 4], [5, 6]], 250)` returns:

```
[[255, 254], [255, 255]]
```

You may assume that the provided pixel_grid contains at least one row and one column and that each row will contain the same number of pixels. Do NOT change the original pixel_grid. Your function should return a new pixel_grid.

You should call `adjusted_pixel` (the function you wrote for part a) in your solution.

```
def adjusted_grid(pixel_grid, amount):
    # Write your code here

    new_grid = []
    rows = len(pixel_grid)
    cols = len(pixel_grid[0])
    for row in range(rows):
        new_row = []
        for col in range(cols):
            new_row.append(adjust_pixel(pixel_grid[row][col], amount))
        new_grid.append(new_row)
    return new_grid
```